# Complex Engineering Problem Final Design and Assessment (EE-313)

**Project Title:**

Design and Implementation of a Microcontroller-Based Speed Control System for a DC Motor Using a Three-Phase AC Supply.

**Project Team**

| S.No. | Name | Roll No. | Section |
|---|---|---|---|
| 1 | Muhammad Taha* | EE-23319 | D |
| 2 | Ali Nasir | EE-23328 | D |

**Summary of Progress:**

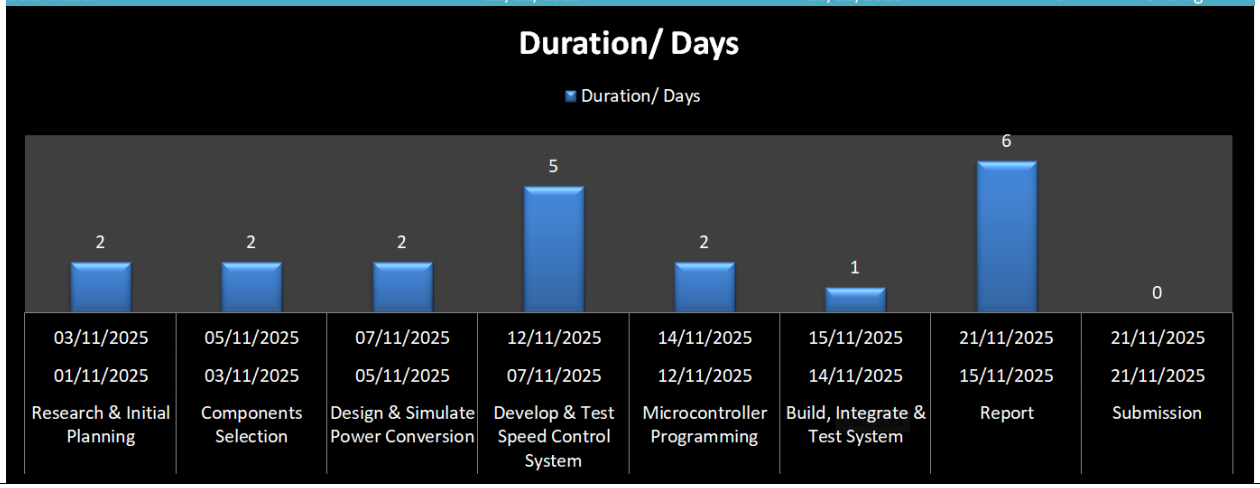
The hardware setup includes a 3-phase half-wave rectifier, buck converter, Arduino, motor driver module, IR encoder sensor, DC encoder motor, and an LCD display. The three-phase AC input is first rectified using the half-wave rectifier, producing pulsating DC. This DC is then fed into the buck converter, which regulates and steps down the voltage to a stable, adjustable DC output suitable for powering the DC motor.

The regulated output of the buck converter is connected to the motor driver's power input terminals. The motor driver then supplies controlled voltage and current to the DC encoder motor while providing electrical isolation for the Arduino. The encoder disc attached to the motor shaft passes through the IR sensor, generating pulses whenever the IR beam is interrupted. These pulses are sent to one of the Arduino's interrupt pins for accurate RPM calculation.

The Arduino generates a PWM signal to the motor driver's control pin to regulate the motor speed based on the reference value. The real-time measured RPM is displayed on a 16x2 LCD connected to the Arduino using digital pins, enabling continuous monitoring of motor performance.

**Task Completed**

| Task | Start | End | Duration/ Days | Status |
|---|---|---|---|---|
| Research & Initial Planning | 01/11/2025 | 03/11/2025 | 2 | Completed |
| Components Selection | 03/11/2025 | 05/11/2025 | 2 | Completed |
| Design & Simulate Power Conversion | 05/11/2025 | 07/11/2025 | 2 | Completed |
| Develop & Test Speed Control System | 07/11/2025 | 12/11/2025 | 5 | Completed |
| Microcontroller Programming | 12/11/2025 | 14/11/2025 | 2 | Completed |
| Build, Integrate & Test System | 14/11/2025 | 15/11/2025 | 1 | Circuit Ready |
| Report | 15/11/2025 | 21/11/2025 | 6 | Ready |
| Submission | 21/11/2025 | 21/11/2025 | 0 | Pending |




**Speed measurement and control of a DC motor**

**Hardware Connectivity**

The hardware system consists of a 3-phase half-wave rectifier, a buck converter, Arduino, IR encoder sensor, motor driver, and the DC encoder motor. The three-phase AC input is first supplied to the half-wave rectifier, where each phase delivers a pulsating DC output. This rectified DC is then fed into the buck converter, which steps down and regu-

lates the voltage to a smooth, adjustable DC level suitable for the DC motor.

The output of the buck converter is connected to the power input terminals of the motor driver. The motor driver then supplies the required current and voltage to the DC encoder motor while isolating the Arduino from high power. The encoder disc on the motor passes through the IR sensor; each interruption of the IR beam produces a pulse. These pulses are fed into an Arduino interrupt pin for accurate speed measurement.

The Arduino generates a PWM control signal to the motor driver's input pin, adjusting the duty cycle to regulate motor speed. The measured RPM is calculated and displayed on an LCD or serial monitor, forming a complete closed-loop speed control system.

**C code for speed measurement/display**

This embedded C firmware, written using the libopencm3 library for an STM32 microcontroller, generates a square wave of a desired frequency and duty cycle on pin **PA9** using **Timer1 in PWM mode**. The program begins by configuring the system clock to 84 MHz, enabling the GPIOA and Timer1 peripheral clocks, and setting PA9 to alternate-function mode with AF1 (TIM1_CH2). Inside the timer1_setup() function, Timer1 is configured for edge-aligned PWM, down-counting mode, continuous operation, and an immediate auto-reload update. A prescaler of 1 divides the 84 MHz APB2 clock by 2 to obtain a 42 MHz timer clock, and the auto-reload register is set to 8400, defining the PWM frequency. The duty cycle is calculated from a macro (duty_cycle = 40%) to produce the output compare value used by channel 2 of Timer1. The code enables PWM mode (PWM1), activates preload and output compare functionality, loads the initial counter and compare values, and finally starts the timer so PA9 outputs a stable PWM waveform. The main() function simply initializes clocks, configures pin PA9, calls the timer setup routine, and then enters an infinite loop where the microcontroller continuously generates the PWM signal.

**Complete Hardware-Software Description**

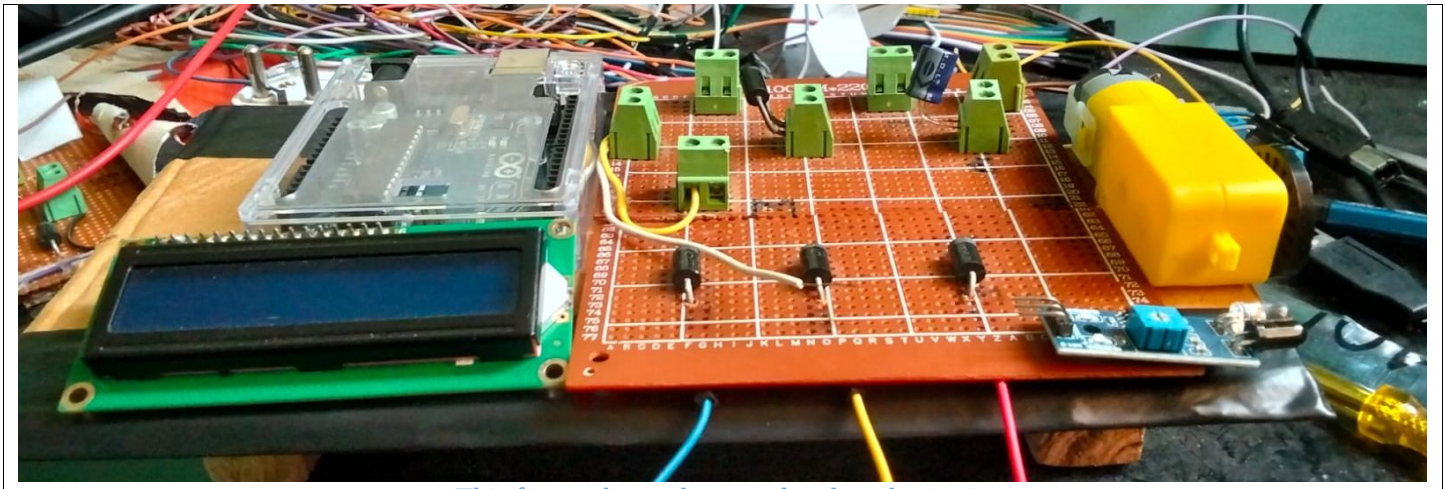
**Hardware**

The hardware system includes a 3-phase half-wave rectifier, buck converter, DC encoder motor, motor driver module, IR encoder sensor, STM32 microcontroller, and a 16x2 LCD display. The three-phase AC is first rectified using the half-wave rectifier to produce pulsating DC, which is then regulated by the buck converter to a smooth, adjustable voltage suitable for the motor. The motor driver receives this regulated DC to safely drive the motor while isolating the microcontroller. The IR encoder sensor detects the rotation of the motor's encoder disc and generates pulses corresponding to shaft movement. These pulses are used for real-time speed measurement and feedback. The measured RPM is displayed on the LCD for monitoring purposes.

**Software**

The software is developed in VS Code for the STM32 microcontroller. It reads pulses from the IR encoder using interrupt pins to calculate the motor's real-time RPM. A PWM signal is generated by the STM32 and sent to the motor driver to control motor speed. A closed-loop feedback algorithm adjusts the PWM duty cycle to maintain the desired speed. The software also updates the LCD continuously to display the measured RPM, providing real-time monitoring and control.

**Results**



This figure shows the completed ready cicuit.

**Final Bill Of Materials (BOM)**

**Complete actual BOM**

| S.No. | Equipment/Component Name | Estimated Cost (PKR) | Reference of Cost Estimate |
|---|---|---|---|
| 1 | 7 Diodes | 64/- (8 each ) | Local Shop |
| 2 | Capacitor | 30/- | Local Shop |
| 3 | 3 Screw connector | 90/- (15 each) | Local Shop |
| 4 | Inductor | 240/- | Local Shop |
| 5 | Vero board | 150/- | Local Shop |
| 6 | Arduino UNO | 1400/- | Local Shop |
| 7 | Stm-32 F40 | 690/- | Local Shop |
| 8 | St-Link V2 | 450/- | Local Shop |
| 9 | Solder iron | 550/- | Local Shop |
| 10 | DC motor | 350/- | Local Shop |
| 11 | IR sensor | 100/- | Local Shop |
| | **Total Cost:** | **4,114/-** | **Local Shop** |

**Risk Assessment and Mitigation**

**Risk Mitigation**

**Identified Risks:**

1. Overheating of DC motor due to continuous high load or voltage fluctuations.
2. Incorrect PWM signal or software bug causing unstable motor operation.
3. IR sensor misalignment or failure leading to inaccurate speed measurement.
4. Electrical hazards from the 3-phase rectifier, buck converter, or high-current connections.
5. LCD display failure affecting RPM monitoring.

**Mitigation Strategies:**

- **Risk Avoidance:** Ensure proper insulation, secure wiring, and correct mounting of the IR sensor to prevent misalignment or short circuits.
- **Risk Reduction:** Use the motor driver to handle high currents safely; include heat sinks or cooling for the motor and buck converter. Implement software safeguards like maximum PWM limits.
- **Risk Monitoring and Control:** Continuously monitor motor temperature and voltage levels; use serial monitor and LCD to observe RPM in real time.
- **Risk Acceptance:** Minor display glitches or temporary pulse miscounts are tolerated as they do not harm hardware.
- **Team Coordination:** Tasks were reviewed collaboratively to ensure proper handling of electrical and software risks.

These strategies collectively ensure safe, reliable, and accurate motor operation during testing and demonstration.

**References**


[1] M. H. Rashid, *Power Electronics: Circuits, Devices and Applications*, 4th ed., Pearson, 2014.
[2] R. Krishnan, *Electric Motor Drives: Modeling, Analysis, and Control*, 2nd ed., Prentice Hall, 2001.
[3] A. K. Sawhney, *Electrical Machines*, 7th ed., Delhi: Dhanpat Rai & Co., 2017.
[4] J. F. Witte, *Microcontroller-Based DC Motor Control*, Springer, 2018.
[5] STM32F4 Reference Manual, STMicroelectronics, 2020. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html
[6] Arduino Documentation, Arduino.cc. [Online]. Available: https://www.arduino.cc/en/Reference/HomePage
[7] "Pulse Width Modulation (PWM) for DC Motor Speed Control," *All About Circuits*, 2023. [Online]. Available: https://www.allaboutcircuits.com/projects/pwm-for-dc-motor-speed-control/
[8] H. A. Toliyat and G. B. Kliman, *Handbook of Electric Motors*, 2nd ed., CRC Press, 2004.

## Attachments:

**Arduino Code For Display RPM**

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
volatile int pulseCount = 0;
unsigned long lastTime = 0;
float rpm = 0;
const int sensorPin = 2;          // SENSOR CONNECTED TO D2 (INT0)
const int pulsesPerRevolution = 1;
const unsigned long interval = 10000; // 10 seconds
void setup() {
  pinMode(sensorPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(sensorPin), countPulse, FALLING);
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("COMPLEX ENG");   // First line
lcd.setCursor(0, 1);
lcd.print("PROBLEM");       // First line continued / second word
delay(2000);                // Show first message
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("INSTRUCTOR:");
lcd.setCursor(0, 1);
lcd.print("SIR Javed");
delay(2000);                // Show instructor
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("GROUP MEMBER:");
lcd.setCursor(0, 1);
lcd.print("M.TAHA & ALI");
delay(2000);
  lastTime = millis();
}
void loop() {
  unsigned long currentTime = millis();
  if (currentTime - lastTime >= interval) {
    noInterrupts();
    int count = pulseCount;
    pulseCount = 0;
    interrupts();
    rpm = (count * 60.0) / pulsesPerRevolution / (interval / 1000.0);
    Serial.print("RPM: ");
    Serial.println(rpm, 2);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Motor RPM:");
    lcd.setCursor(0, 1);
    lcd.print(rpm, 2);
    lastTime = currentTime;
  }
}
void countPulse() {
  pulseCount++;
}
```

## STM Code For Measure Duty Cycle

```
#include <libopencm3/stm32/rcc.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/timer.h>
#define duty_cycle  40
#define reg_val     (uint32_t)(((float)duty_cycle/100.0)*8400)
void timer1_setup()
{
    rcc_periph_clock_enable(RCC_TIM1);
    timer_set_alignment(TIM1,TIM_CR1_CMS_EDGE);
    timer_direction_down(TIM1);
    timer_continuous_mode(TIM1);
    timer_disable_preload(TIM1);
    timer_set_prescaler(TIM1,1);
    timer_set_period(TIM1,8400);
    timer_set_oc_mode(TIM1,TIM_OC2,TIM_OCM_PWM1);
    timer_enable_oc_preload(TIM1,TIM_OC2);
    timer_enable_oc_output(TIM1,TIM_OC2);
    timer_enable_break_automatic_output(TIM1);
    timer_set_oc_value(TIM1,TIM_OC2,reg_val);
    timer_generate_event(TIM1,TIM_EGR_UG);
    timer_set_counter(TIM1,0);
    timer_enable_counter(TIM1);
}
int main(void)
{
    rcc_clock_setup_pll(&rcc_hse_25mhz_3v3[RCC_CLOCK_3V3_84MHZ]);
    rcc_periph_clock_enable(RCC_GPIOA);
   gpio_mode_setup(GPIOA,GPIO_MODE_AF,
        GPIO_PUPD_PULLDOWN,GPIO9);
    gpio_set_output_options(GPIOA,GPIO_OTYPE_PP,
        GPIO_OSPEED_25MHZ,GPIO9);
    gpio_set_af(GPIOA,GPIO_AF1,GPIO9);
    timer1_setup();
    while(1)
    {
        __asm__("nop");
    }
    return 0;
}
```

| Skill(s) to be assessed | Extent of Achievement | | | |
|---|---|---|---|---|
| | <50% | 50%-65% | 65%-80% | 80%-100% |
| **Application of theoretical knowledge for analysis** | Unable to relate and recall any theory | Partially relates and recalls the theory | Recalls theoretical knowledge and relates it to the task without major mistakes | Recalls theoretical knowledge and relates it to the task with no mistake |
| **Updates in Resource identification and allocation, task identification and scheduling, risk assessment and management** | No updates or revision | ---------- | ---------- | Reasonable updates and revision presented |
| **Realisation of design, testing and validation** | Unable to synthesize and also unable to test/validate | Able to synthesize partially and able to test/validate OR Able to synthesize but partially able to test/validate | Able to synthesize and test/validate without major mistakes | Able to synthesize and test/validate without any mistake |
| **Reporting hardware testing results in verbal and graphical manner** | No results reported | Results reported but missing major deliverables | Results reported with minor issues | Results reported, meeting all deliverables |

| Skill(s) to be assessed | Extent of Achievement | | | |
|---|---|---|---|---|
| | <50% | 50%-65% | 65%-80% | 80%-100% |
| **Application of theoretical knowledge for analysis** | Unable to relate and recall any theory | Partially relates and recalls the theory | Recalls theoretical knowledge and relates it to the task without major mistakes | Recalls theoretical knowledge and relates it to the task with no mistake |
| **Updates in Resource identification and allocation, task identification and scheduling, risk assessment and management** | No updates or revision | ---------- | ---------- | Reasonable updates and revision presented |
| **Realisation of design, testing and validation** | Unable to synthesize and also unable to test/validate | Able to synthesize partially and able to test/validate OR Able to synthesize but partially able to test/validate | Able to synthesize and test/validate without major mistakes | Able to synthesize and test/validate without any mistake |
| **Reporting hardware testing results in verbal and graphical manner** | No results reported | Results reported but missing major deliverables | Results reported with minor issues | Results reported, meeting all deliverables |